

4

LABORATORY FOR
COMPUTER SCIENCE



MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

AD-A210 829

MIT/LCS/TM-383

A SPECIAL CASE OF SECOND-ORDER STRICTNESS ANALYSIS

DTIC
ELECTE
AUG 02 1989
S D CS D

Paul P. Wang

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

February 1989

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TM-383			5. MONITORING ORGANIZATION REPORT NUMBER(S) N00014-83-K-0125		
6a. NAME OF PERFORMING ORGANIZATION MIT Laboratory for Computer Science		6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Department of Navy		
6c. ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139			7b. ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22217			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
					WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) <u>A Special Case of Second-Order Strictness Analysis</u>					
12. PERSONAL AUTHOR(S) Wang, Paul P.					
13a. TYPE OF REPORT Technical		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) 1989 February	
15. PAGE COUNT 19					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
			strictness analysis, abstract interpretation, termination properties		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) A function declaration is called strict in one of its formal parameters if, in all calls to the function, either the actual corresponding parameter is evaluated, or the call does not terminate. Approximating strictness analysis with abstract interpretation reduces to the evaluation of recursive monotone Boolean functions. This evaluation problem is complete in deterministic exponential time when the functions are declared with only base-type formal parameters, and is deterministic super-exponential time hard when functions are formal parameters. However, by coarsening the strictness analysis approximation to be conjunctive, the first-order case can be completed in linear time. This paper will show that the second-order case is NP-hard.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL Judv Little, Publications Coordinator			22b. TELEPHONE (Include Area Code) (617) 253-5894		22c. OFFICE SYMBOL



A Special Case of Second-Order Strictness Analysis*

Paul P. Wang[†]

MIT Laboratory for Computer Science

February 15, 1989

Accession For		
NTIS	CRA&I	<input checked="" type="checkbox"/>
DTIC	TAB	<input type="checkbox"/>
Unannounced		<input type="checkbox"/>
Justification		
By		
Distribution		
Availability Codes		
Dist	Avail and/or Special	
A-1		

Abstract

A function declaration is called *strict* in one of its formal parameters if, in all calls to the function, either the actual corresponding parameter is evaluated, or the call does not terminate. Approximating *strictness analysis* with *abstract interpretation* reduces to the evaluation of recursive monotone Boolean functions. This evaluation problem is complete in deterministic exponential time when the functions are declared with only base-type formal parameters, and is deterministic super-exponential time hard when functions are formal parameters. However, by coarsening the strictness analysis approximation to be *conjunctive*, the first-order case can be completed in linear time. This paper will show that the second-order case is *NP*-hard.

*This memo is a report based on the author's Senior Thesis, submitted in June, 1988 to the Department of Electrical Engineering and Computer Science, under the supervision of Prof. Albert R. Meyer.

[†]This research was supported in part by a NSF Graduate Fellowship, NSF Grant 98477-8511190-CCR, and DARPA Grant N00014-83-K-0125.

1 Introduction

Call-by-need evaluation strategies are used in the semantics of many modern programming languages. The advantage of a call-by-need interpreter, as opposed to one using *call-by-value* strategies, is that normal-order evaluation (which corresponds to call-by-need) will terminate in many cases where applicative-order evaluation (call-by-value) will not. However, in cases where both strategies will terminate, a call-by-value interpreter is generally regarded as being more efficient due to the overhead associated with call-by-need's "delayed" representation of objects.

Strictness analysis is a way of reducing such overhead. A function declaration is called *strict* in one of its formal parameters if, in all calls to the function, either the corresponding actual parameter is evaluated or the call does not terminate on the usual call-by-need style interpreter. Strictness analysis determines the termination properties of the function being analyzed. The results of such analysis have been used to optimize both parallel and sequential implementations of computer languages. If a function is strict in all its formal parameters, then converting a call-by-need interpreter to use a call-by-value strategy for evaluating the function's arguments will not change the results of the program.

Recently, several papers have offered a method, namely *abstract interpretation*, to infer an approximation of strictness of a function. More specifically, all program results can be abstracted to the domain $\{\perp, \top\}$, where $\perp \sqsubset \top$. \perp is interpreted as "the program does not terminate". \top is interpreted as "the program might or might not terminate". The operators \sqcap and \sqcup compute the minimum and maximum of two arguments. All base-type constants are mapped to \top because they always terminate. Also, strict functions such as $+$ (integer addition) and \wedge (Boolean *and* operation) are mapped to $\lambda xy.x \sqcap y$ since they diverge if either argument diverges. Conditionals are mapped to $\lambda pca.p \sqcap (c \sqcup a)$. See [6, 9, 1] for details.

The above domain $\{\perp, \top\}$ with the operators \sqcap and \sqcup can also be viewed as the Boolean values "true" and "false" with the Boolean operators "and" and "or".

Because many programs that are analyzed include recursion, this approximation of strictness analysis reduces to the evaluation of recursive monotone Boolean functions (RMBF's) [6, 9, 1]. Furthermore, since many program sub-routines are allowed to be declared simultaneously and thus can "recursively

call each other”, such an approximation of strictness analysis must reduce to the evaluation of schemes representing sets of simultaneously declared RMBF’s. This paper will define such schemes (called recursive monotone schemes or RMS’s) as well as evaluation decidability problems for RMS’s.

Recent work has shown that first-order strictness analysis approximation using abstract interpretation (i.e. the functions to be analyzed are declared with only base-type formal parameters) is complete in deterministic exponential time [3, 6]. Furthermore, higher-order strictness analysis approximation (i.e. the functions to be analyzed are declared with functions as formal parameters) is deterministic “super-exponential” time hard [3, 12].

There have been many attempts to reduce the average time efficiency of strictness analysis algorithms to more acceptable levels. One way to do this is to develop algorithms which take less time “in the average case” [5, 2]. Another way is to modify strictness analysis by restricting the amount of information that we can extract through abstract interpretation [11, 7]. This paper discusses the complexity of strictness analysis when the analysis is coarsened to treat the conditional as though it were an ordinary algebraic function known only to be strict in its first argument (i.e. conditionals are mapped to $\lambda pca.p$). We refer to this as *conjunctive* strictness analysis.

Conjunctive strictness analysis reduces to the evaluation of recursive monotone schemes whose syntactic representations do not contain any occurrences of “ \sqcup ” symbols. We refer to these as recursive conjunctive monotone schemes (RCMS’s). Conjunctive strictness analysis in the first-order case turns out to be a linear-time problem with respect to the length of the analyzed function’s syntactic definition [7]. However, in the second-order case (i.e. when first-order functions are formal parameters), conjunctive strictness analysis is *NP*-hard.

This paper assumes that the reader is familiar with the elementary definitions of domain theory and computation theory.

2 Formal Definitions and Notation

In order to design a scheme that represents a set of simultaneously declared recursive monotone Boolean functions, we first formalize the domain of abstract interpretation (Section 2.1). We next introduce “types” and “terms” to formalize our syntax (Section 2.2). After defining recursive monotone

schemes (RMS's) as representing sets of simultaneously declared RMBF's (Section 2.3), we then assign meaning to these schemes (Section 2.4).

2.1 The Abstract Interpretation Domain

Definition 2.1 Let $\mathbf{2}$ be the domain $\{\top, \perp\}$ where $\perp \sqsubset \top$. For partial orders C, D , let $C \rightarrow D$ denote the monotone function space. Let $C \times D$ denote the cartesian product of C and D . We will use the standard "pointwise" partial orders for the sets $C \times D$ and $C \rightarrow D$. Refer to [10] for details.

We will denote the least element in the set C (if it exists) by \perp_C . (Note that $\perp_{\mathbf{2}} = \perp$.)

2.2 Types and Terms

In order to define our recursive monotone schemes, we must first introduce the concept of *types*. An object's type determines the "functional behavior" of what the object intuitively represents.

Definition 2.2 The set of types derived over the base type b can be defined by the following BNF grammar:

$$\begin{aligned}\tau &::= b \mid \langle \tau_{list} \rangle \rightarrow \tau \\ \tau_{list} &::= \tau \mid \tau_{list} \times \tau\end{aligned}$$

Objects of type b represent the values $\{\top, \perp\}$, the symbol \times represents cartesian product, and the symbol \rightarrow represents monotone function space.

We define depth of types as:

Definition 2.3 The depth of type τ is defined as follows:

- $depth(b) = 0$.
- $depth(\langle \tau_1 \times \tau_2 \times \dots \times \tau_n \rangle \rightarrow \tau_{n+1}) = 1 + \max_{1 \leq i \leq n+1} [depth(\tau_i)]$

Let " x^τ " denote a typed variable symbol of type τ . Let \perp^b and \top^b be constant symbols.

We will now introduce the concepts of preterms, type-statements, and terms.

Definition 2.4 Preterms, e , are given by the following BNF grammar:

$$\begin{aligned}
e &::= \text{var} \mid \perp^b \mid \top^b \mid e \sqcup e \mid e \sqcap e \mid e(e_{list}) \mid \\
&\quad \lambda \text{var}_{list}.e \\
\text{var} &::= \mathbf{x}_0^\tau \mid \mathbf{x}_1^\tau \mid \mathbf{x}_2^\tau \mid \dots \quad (\tau \text{ is a type}) \\
\text{var}_{list} &::= \text{var} \mid \text{var}_{list} \text{ var} \\
e_{list} &::= e \mid e_{list}, e
\end{aligned}$$

Definition 2.5 Type-Statements are pairs $e : \tau$, where e is a pre-term and τ is a type, defined inductively as follows:

- $\perp^b : b$ and $\top^b : b$ are type-statements.
- $\mathbf{x}^\tau : \tau$ is a type-statement.
- $$\frac{e_1 : \tau, \quad e_2 : \tau}{e_1 \sqcap e_2 : \tau}$$
- $$\frac{e_1 : \tau, \quad e_2 : \tau}{e_1 \sqcup e_2 : \tau}$$
- $$\frac{e : \langle \tau_1 \times \tau_2 \times \dots \times \tau_n \rangle \rightarrow \tau, \quad e_i : \tau_i \quad (1 \leq i \leq n)}{e(e_1, e_2, \dots, e_n) : \tau}$$
- $$\frac{e : \tau}{\lambda \mathbf{y}_1^{\tau_1} \mathbf{y}_2^{\tau_2} \dots \mathbf{y}_n^{\tau_n}. e : \langle \tau_1 \times \tau_2 \times \dots \times \tau_n \rangle \rightarrow \tau}$$

Definition 2.6 e is a term iff $e : \tau$ is a type-statement for some τ . A term is conjunctive iff it does not contain any occurrences of " \sqcup ".

Note that τ is unique if it exists.

Definition 2.7 The depth of a term, e , is defined inductively as follows:

- $\text{depth}(\perp^b) = \text{depth}(\top^b) = 0$
- $\text{depth}(\mathbf{x}^\tau) = \text{depth}(\tau)$

- $depth(e_1 \sqcap e_2) = \max(depth(e_1), depth(e_2))$
- $depth(e_1 \sqcup e_2) = \max(depth(e_1), depth(e_2))$
- $depth(e_0(e_1, e_2, \dots, e_n)) = \max_{0 \leq i \leq n}(depth(e_i))$
- $depth(\lambda y_1^{\tau_1} y_2^{\tau_2} \dots y_n^{\tau_n}. e) = \max_{0 \leq i \leq n}(1 + depth(\tau_i), depth(e))$,
where $e : \tau_0$.

2.3 Recursive Monotone Schemes

Definition 2.8 A recursive monotone scheme (RMS), S , is an n -tuple $S = ((x_1^{\tau_1}, e_1), (x_2^{\tau_2}, e_2), \dots, (x_n^{\tau_n}, e_n))$, where $e_i : \tau_i$ ($1 \leq i \leq n$) and $FV(e_i) \subseteq \{x_1^{\tau_1}, x_2^{\tau_2}, \dots, x_n^{\tau_n}\}$.

An RMS $S = ((x_1^{\tau_1}, e_1), (x_2^{\tau_2}, e_2), \dots, (x_n^{\tau_n}, e_n))$ represents the simultaneous declaration of n recursive monotone Boolean functions. The set of typed symbols $\{x_1^{\tau_1}, \dots, x_n^{\tau_n}\}$ represent the declared functions, and each term e_i represents the declaration of the function $x_i^{\tau_i}$.

An example of an RMS would be the following:

Example 2.1 Consider the following declaration:

$$\begin{aligned} \text{letrec } f(x, y) &= x \vee g(h(x \wedge y, y), h) \\ \text{and } g(x, y) &= y(x, x) \vee x \\ \text{and } h(x, y) &= x \wedge (y \vee f(y, x)) \end{aligned}$$

The RMS $S = ((x_1^{\tau_1}, e_1), (x_2^{\tau_2}, e_2), (x_3^{\tau_3}, e_3))$ will represent the above declaration, where:

$$\begin{aligned} e_1 &::= \lambda y_1^b y_2^b. [y_1^b \sqcup x_2^{\tau_2}(x_3^{\tau_3}(y_1^b \sqcap y_2^b, y_2^b), x_3^{\tau_3})] \\ e_2 &::= \lambda y_1^b y_2^{b^2 \rightarrow b}. [y_2^{b^2 \rightarrow b}(y_1^b, y_1^b) \sqcup y_1^b] \\ e_3 &::= \lambda y_1^b y_2^b. [y_1^b \sqcap (y_2^b \sqcup x_1^{\tau_1}(y_2^b, y_1^b))] \\ \tau_1 &= b^2 \rightarrow b \\ \tau_2 &= \langle b \times (b^2 \rightarrow b) \rangle \rightarrow b \\ \tau_3 &= b^2 \rightarrow b \quad \square \end{aligned}$$

Definition 2.9 Let the depth of the RMS $S = ((x_1^{\tau_1}, e_1), \dots, (x_n^{\tau_n}, e_n))$ be:

$$\text{depth}(S) = \max_{1 \leq i \leq n} (\text{depth}(e_i))$$

2.4 Interpretations

The purpose of types is to denote the “functional behavior” of what each typed object is representing. Let $\llbracket \cdot \rrbracket$ denote a “type interpreter” that assigns a set of functions to each type derivable under our definitions.

- $\llbracket b \rrbracket = 2$.
- Let $\tau = \langle \tau_1 \times \tau_2 \times \dots \times \tau_n \rangle \rightarrow \tau_{n+1}$.
 $\llbracket \tau \rrbracket = \langle \llbracket \tau_1 \rrbracket \times \llbracket \tau_2 \rrbracket \times \dots \times \llbracket \tau_n \rrbracket \rangle \rightarrow \llbracket \tau_{n+1} \rrbracket$.

We will now introduce the idea of assigning (semantic) values to (syntactic) terms. Let an environment, ρ , represent a mapping from typed variables x^τ to elements in $\llbracket \tau \rrbracket$. Then for any term e , where $e : \tau$, we can assign an element in $\llbracket \tau \rrbracket$ to e . Let $\llbracket e \rrbracket \rho$ denote that element and be inductively defined as follows:

- $\llbracket \perp^h \rrbracket \rho = \perp$ and $\llbracket \top^h \rrbracket \rho = \top$.
- $\llbracket x^\tau \rrbracket \rho = \rho(x^\tau)$.
- $\llbracket e(e_1, e_2, \dots, e_n) \rrbracket \rho = \llbracket e \rrbracket \rho (\llbracket e_1 \rrbracket \rho, \llbracket e_2 \rrbracket \rho, \dots, \llbracket e_n \rrbracket \rho)$.
- $\llbracket e_1 \sqcap e_2 \rrbracket \rho = \llbracket e_1 \rrbracket \rho \wedge \llbracket e_2 \rrbracket \rho$.

where \wedge is the pointwise glb operator.

- $\llbracket e_1 \sqcup e_2 \rrbracket \rho = \llbracket e_1 \rrbracket \rho \vee \llbracket e_2 \rrbracket \rho$.

where \vee is the pointwise lub operator.

- $\llbracket \lambda(x_1^{\tau_1} x_2^{\tau_2} \dots x_n^{\tau_n}). e \rrbracket \rho = f \in I[\langle \tau_1 \times \dots \times \tau_n \rangle \rightarrow \tau]$

where $e : \tau$ and $f(d_1, d_2, \dots, d_n) = \llbracket e \rrbracket \rho[x_i^{\tau_i} \mapsto d_i]$ ($1 \leq i \leq n$, $d_i \in \llbracket \tau_i \rrbracket$). $\rho[x_i^{\tau_i} \mapsto d_i]$ denotes the environment obtained from ρ by letting $x_i^{\tau_i}$ be mapped to d_i .

Let RMS $S = ((x_1^{\tau_1}, e_1), \dots, (x_n^{\tau_n}, e_n))$. We would like to define an environment ρ_S that will assign to each typed symbol $x_i^{\tau_i}$ ($1 \leq i \leq n$) the element of $\llbracket \tau_i \rrbracket$, such that $\rho(x_i^{\tau_i})$ ($1 \leq i \leq n$) are the least fixed points of the following set of equations:

$$\begin{aligned} x_1^{\tau_1} &= e_1 \\ x_2^{\tau_2} &= e_2 \\ &\vdots \\ x_n^{\tau_n} &= e_n \end{aligned}$$

There are many algorithms for generating ρ_S [13].

Thus for any RMS $S = ((x_1^{\tau_1}, e_1), \dots, (x_n^{\tau_n}, e_n))$ and any well-typed term ϵ , such that $c : \tau$ and $FV(\epsilon) \subseteq \{x_1^{\tau_1}, x_2^{\tau_2}, \dots, x_n^{\tau_n}\}$, we can assign to ϵ an element in $\llbracket \tau \rrbracket$ by computing $\llbracket \epsilon \rrbracket_{\rho_S}$. This is the value of the term ϵ when its “function calls” are interpreted as within the scope of the declarations represented by S .

2.5 Problems

In this section, we will formalize some interesting decidability problems relevant to RMS's and strictness analysis.

Definition 2.10 *K*th-Order RMS Evaluation Problem (kE):

$S \in kE$ iff:

- $S = ((x_1^{\tau_1}, e_1), \dots, (x_n^{\tau_n}, e_n))$ is an RMS with $\text{depth}(S) \leq k$ and $\tau_n = b$.
- $\rho_S(x_n^b) = \top$

Theorem 2.1 *The 1E evaluation problem is complete in deterministic exponential time [3, 6].*

Theorem 2.2 *In general, as the depth of the RMS increases, the complexity of its evaluation decidability problem grows as a composition of exponentials [3, 12].*

3 Improving the Running Time

The results of Theorems 2.1 and 2.2 suggest that the actual implementation of strictness analysis is intractable due to its deterministic exponential (and in higher-order cases “super-exponential”) time behavior. However, there are ways of improving the running time. One approach is to devise algorithms whose “average case” behavior is significantly better than exponential (or super-exponential) time. Examples of such methods are Frontier Analysis and Pending Analysis. See [5, 2] for details.

Another approach is to improve the running time of strictness analysis by restricting the amount of information that can be extracted from abstract interpretation. Abstract interpretation approximates conditionals by mapping them to $\lambda pca.p \sqcap (c \sqcup a)$. However, by “coarsening” abstract interpretation to treat conditionals as though they were ordinary algebraic functions that were known to be strict only in the first argument (i.e. map conditionals to $\lambda pca.p$), we get a “coarser” approximation of strictness analysis. However, such a sacrifice in accuracy could yield a significant improvement in running time. We refer to this version of strictness analysis as *conjunctive* strictness analysis.

Conjunctive strictness analysis reduces to the evaluation of RMS’s whose syntactic definition does not contain any occurrences of the “ \sqcup ” symbol. We refer to such a RMS as a *recursive conjunctive monotone scheme* or RCMS. We define the k -th Order RCMS Evaluation Problem (kEC) as the special case of the kE evaluation problem where the RMS evaluated is also an RCMS.

With minor corrections and revisions on the work by [7], it can be shown that the 1EC decidability problem takes deterministic linear time. This improvement in running time (i.e. from deterministic exponential to linear time) is not as impressive as it may seem, since the “coarsened” strictness analysis significantly limits the amount of information that can be approximated. However, it would be interesting to see if the 2EC decidability problem remains tractable (i.e. remains in polynomial time), if it becomes exponential in complexity, or perhaps stays super-exponential. The next section will show that 2EC is, in fact, *NP*-hard.

4 Second-Order Conjunctive Strictness

The 2EC decidability problem is *NP*-hard because there exists a polynomial time many-one reduction from the GSAT decidability problem. We first formally define the GSAT problem (Section 4.1). Then we show a poly-time many-one reduction from GSAT to 2EC (Section 4.2).

4.1 GSAT

The GSAT decidability problem (the satisfiability problem for propositional calculus) is a commonly known *NP*-complete problem. This paper will assume the reader's knowledge of the syntax of propositional calculus formulas as well as the concepts of Boolean variables, truth-assignments, verifying and satisfiability. For more details, please refer to [4, 8].

Definition 4.1 GSAT Problem:

$GSAT = \{\phi(p_1, p_2, \dots, p_n) \mid \phi \text{ is a propositional calculus formula with Boolean variables } p_1, p_2, \dots, p_n \text{ under the operators } \neg, \wedge, \text{ and } \vee, \text{ such that } \phi \text{ has a satisfying truth-assignment}\}$

We will define a truth-assignment, A , for the Boolean variables of ϕ as a mapping from the Boolean variables of ϕ to the set $\{1, 0\}$, where 1 and 0 denote the Boolean values *true* and *false*, respectively.

The GSAT decidability problem is *NP*-complete [4, 8].

4.2 GSAT and 2EC

In this section, we show a poly-time many-one reduction from GSAT to 2EC. Before we do this we need to present the following Lemma:

Lemma 4.1 For any n bit vector $\mathbf{a} = (a_{n-1}, a_{n-2}, \dots, a_0)$, let $\text{bin}(\mathbf{a})$ be the integer k ($0 \leq k < 2^n$) with the n -bit representation \mathbf{a} . One can construct for any j ($0 \leq j \leq n-1$), a propositional calculus formula $\phi_j^n(a_{n-1}, a_{n-2}, \dots, a_0)$ such that:

$$\text{bin}(\phi_j^n(\mathbf{a})) = (\text{bin}(\mathbf{a}) + 1) \bmod 2^n$$

In fact, it is not hard to see that ϕ_j^n need only have $O(n)$ connectives and can be constructed, given n and j , in time polynomial in n .

Example 4.1 *The following formulas $\phi_0^3, \phi_1^3, \phi_2^3$ are defined by:*

$$\begin{aligned}\phi_0^3 &::= \neg a_0 \\ \phi_1^3 &::= (\neg a_0 \vee \neg a_1) \wedge (a_0 \vee a_1) \\ \phi_2^3 &::= (\neg a_0 \vee \neg a_1 \vee \neg a_2) \wedge ((a_0 \wedge a_1) \vee a_2) \quad \square\end{aligned}$$

We now introduce a restricted class of propositional calculus formulas, called “negation-pushed” formulas. A negation-pushed formula is a propositional calculus formula, where all occurrences of the “ \neg ” symbol immediately precede a Boolean variable. Note that every propositional calculus formula can be transformed in linear time into a negation-pushed formula by repeated use of DeMorgan’s Law and the identity $\neg\neg p \equiv p$.

Before we go any further, we should make the following comments on notation to be used in this section:

- Let I denote the identity function of the domain **2**.
- Let T denote the constant function \top from **2** to **2**.

We now define f , a mapping from propositional calculus formulas to conjunctive terms of type $b \rightarrow b$. The difficulty in such a mapping is how to “simulate” the Boolean *or* operator in conjunctive terms, since such terms cannot contain any occurrences of the “ \sqcup ” symbol. The way around such a difficulty is to represent Boolean values with first-order functions and the Boolean *or* operator with the composition of first-order functions. The main ideas of the mapping f can be summarized below:

- “simulate” the Boolean value 0 with the first-order function I .
- “simulate” 1 with T .
- “simulate” Boolean *and* with \sqcap , i.e. pointwise glb.

- “simulate” Boolean *or* with the composition of two first-order functions.

The basis for these ideas are formalized in the following Lemma:

Lemma 4.2 *Let $\alpha(1) = T$, $\alpha(0) = I$ and $a_1, a_2 \in \{1, 0\}$. Then:*

$$\begin{aligned}\alpha(a_1) \sqcap \alpha(a_2) &= \alpha(a_1 \wedge a_2) \\ \alpha(a_1) \circ \alpha(a_2) &= \alpha(a_1 \vee a_2)\end{aligned}$$

where \wedge and \vee represent Boolean *and* or *respectively*, \sqcap represents point-wise *glb*, and \circ represents *composition*.

With the above Lemma in mind, we now formally define the mapping f . Given a propositional calculus formula $\phi(p_{n-1}, p_{n-2}, \dots, p_0)$, define $f(\phi)$ as follows:

- $FV(f(\phi)) \subseteq \{y_{n-1}^b, z_{n-1}^b, y_{n-2}^b, z_{n-2}^b, \dots, y_0^b, z_0^b\}$
- 1. Transform ϕ to its negation-pushed form ϕ' .
- 2. Transform ϕ' into a conjunctive term of type $b \rightarrow b$ using a mapping g defined inductively as follows:
 - $g(p_i) ::= y_i^{b \rightarrow b}$
 - $g(\neg p_i) ::= z_i^{b \rightarrow b}$
 - $g(\phi_1 \wedge \phi_2) ::= g(\phi_1) \sqcap g(\phi_2)$
 - $g(\phi_1 \vee \phi_2) ::= g(\phi_1) \circ g(\phi_2)$

where, for clarity purposes, the above is an abbreviation for:
 $g(\phi_1 \vee \phi_2) ::= \lambda x^b. [\{g(\phi_1)\}(\{g(\phi_2)\}(x^b))]$

f can obviously be computed in $O(|\phi|)$ time.

Example 4.2 *Consider the following propositional calculus formula ϕ :*

$$\phi ::= p_0 \vee \neg(p_1 \wedge \neg p_2)$$

Then $f(\phi)$ is defined by:

$$f(\phi) ::= y_0^{b \rightarrow b} \circ z_1^{b \rightarrow b} \circ y_2^{b \rightarrow b} \quad \square$$

Before we make any formal conclusions about f , we must introduce the concept of a *well-formed environment*.

Definition 4.2 A well-formed environment, ρ , is an environment such that, $\forall i, \rho(y_i^{b \rightarrow b}) \in \{T, I\}$ and:

$$\rho(z_i^{b \rightarrow b}) = \begin{cases} I, & \text{if } \rho(y_i^{b \rightarrow b}) = T \\ T, & \text{otherwise} \end{cases}$$

We can now make the following conclusion about f :

Lemma 4.3 Let $\phi(p_{n-1}, p_{n-2}, \dots, p_0)$ be a Boolean formula and A be a truth-assignment for the variables of ϕ . Then:

- If A verifies ϕ , then $\llbracket f(\phi) \rrbracket \rho_A = T$
- If A does not verify ϕ , then $\llbracket f(\phi) \rrbracket \rho_A = I$

where ρ_A is the well-formed environment defined by:

$$\rho_A(y_i^{b \rightarrow b}) = \begin{cases} T, & \text{if } A(p_i) = 1 \\ I, & \text{otherwise} \end{cases}$$

Proof of Lemma 4.3: Can be shown by induction on the length of ϕ using Lemma 4.2. \square

With the above Lemma, we can now show a poly-time many-one reduction from GSAT to 2EC.

Theorem 4.1 GSAT is poly-time many-one reducible to 2EC.

Proof of Theorem 4.1: For any Boolean formula $\phi(p_{n-1}, p_{n-2}, \dots, p_0)$, define $S_\phi = ((x_1^{(b \rightarrow b)^{2n} \rightarrow (b \rightarrow b)}, e_1), (x_2^{b \rightarrow b}, e_2), (x_3^b, e_3))$ to be an RCMS of depth 2, where:

$$\begin{aligned} e_1 &::= \lambda y_{n-1}^{b \rightarrow b} z_{n-1}^{b \rightarrow b} y_{n-2}^{b \rightarrow b} z_{n-2}^{b \rightarrow b} \dots y_0^{b \rightarrow b} z_0^{b \rightarrow b}. \\ &\quad \{f(\phi) \circ x_1^{(b \rightarrow b)^{2n} \rightarrow (b \rightarrow b)}\} \\ &\quad (f(\phi_{n-1}^n), f(\neg \phi_{n-1}^n), f(\phi_{n-2}^n), \dots, f(\neg \phi_0^n)) \\ e_2 &::= x_1^{(b \rightarrow b)^{2n} \rightarrow (b \rightarrow b)} (\lambda x^b . x^b, \lambda x^b . \top^b, \dots, \lambda x^b . x^b, \lambda x^b . \bar{\top}^b) \\ e_3 &::= x_2^{b \rightarrow b} (\perp^b) \end{aligned}$$

where the mapping f is previously defined, and the formulas ϕ_j^n are defined by Lemma 4.1.

By Lemma 4.1 and our semantics defined in Section 2.4:

$$\llbracket x_2^{b \rightarrow b} \rrbracket \rho_{S_\phi} = \llbracket f(\phi) \rrbracket \rho_0 \circ \llbracket f(\phi) \rrbracket \rho_1 \circ \llbracket f(\phi) \rrbracket \rho_2 \circ \cdots \circ \llbracket f(\phi) \rrbracket \rho_{2^n-1} \circ \llbracket x_2^{b \rightarrow b} \rrbracket \rho_{S_\phi}$$

where \circ denotes composition and ρ_k , $k = \text{bin}((a_{n-1}, a_{n-2}, \dots, a_0))$, is the well-formed environment defined by:

$$\rho_k(y_i^{b \rightarrow b}) = \begin{cases} \text{T} & \text{if } a_i = 1 \\ \text{I} & \text{otherwise} \end{cases} \quad (0 \leq i \leq n-1)$$

By Lemma 4.3, if ϕ is satisfiable, then, for some $k < 2^n$, $\llbracket f(\phi) \rrbracket \rho_k = \text{T}$. (For all other $k < 2^n$, $\llbracket f(\phi) \rrbracket \rho_k = \text{I}$.) This will mean that $\llbracket x_2^{b \rightarrow b} \rrbracket \rho_{S_\phi} = \text{T}$. This implies that $\llbracket x_3^b \rrbracket \rho_{S_\phi} = \text{T}$.

Also by Lemma 4.3, if ϕ is not satisfiable, then, for all $k < 2^n$, $\llbracket f(\phi) \rrbracket \rho_k = \text{I}$. Since ρ_{S_ϕ} must specify least-fixpoints, this means that $\llbracket x_2^{b \rightarrow b} \rrbracket \rho_{S_\phi} = \perp_{2 \rightarrow 2}$ (i.e. the constant function \perp from $\mathbf{2}$ to $\mathbf{2}$). This implies that $\llbracket x_3^b \rrbracket \rho_{S_\phi} = \perp$.

Thus $\phi \in \text{GSAT}$ iff $S_\phi \in \text{2EC}$.

We have shown that $f(\phi)$ takes $O(|\phi|)$ time to compute. We have also shown that each ϕ_j^n takes polynomial time to compute and has $O(n)$ connectives. Thus the entire reduction from ϕ to S_ϕ can be done in polynomial time. \square

We can now conclude that the 2EC evaluation problem is *NP*-hard.

Example 4.3 Consider the propositional calculus formula ϕ used in Example 4.2, where:

$$\phi ::= p_0 \vee \neg(p_1 \wedge \neg p_2)$$

By using the above reduction, we construct:

$$S_\phi = ((x_1^{(b \rightarrow b)^b \rightarrow (b \rightarrow b)}, e_1), (x_2^{b \rightarrow b}, e_2), (x_3^b, e_3))$$

where S_ϕ is an RCMS with depth 2 and:

$$\begin{aligned}
e_1 &::= \lambda y_2^{b \rightarrow b} z_2^{b \rightarrow b} y_1^{b \rightarrow b} z_1^{b \rightarrow b} y_0^{b \rightarrow b} z_0^{b \rightarrow b} \cdot \\
&\quad \{f(\phi) \circ x_1^{(b \rightarrow b)^6 \rightarrow (b \rightarrow b)}\} \\
&\quad (f(\phi_2^3), f(\neg \phi_2^3), f(\phi_1^3), f(\neg \phi_1^3), f(\phi_0^3), f(\neg \phi_0^3)) \\
e_2 &::= x_1^{(b \rightarrow b)^6 \rightarrow (b \rightarrow b)} (\lambda x^b . x^b, \lambda x^b . \top^b, \lambda x^b . x^b, \lambda x^b . \top^b, \lambda x^b . x^b, \lambda x^b . \top^b) \\
e_3 &::= x_2^{b \rightarrow b} (\perp^b)
\end{aligned}$$

where mapping f is previously defined and formulas ϕ_k^3 are constructed in Example 4.1. Specifically:

$$\begin{aligned}
f(\phi) &::= y_0^{b \rightarrow b} \circ z_1^{b \rightarrow b} \circ y_2^{b \rightarrow b} \\
f(\phi_0^3) &::= z_0^{b \rightarrow b} \\
f(\neg \phi_0^3) &::= y_0^{b \rightarrow b} \\
f(\phi_1^3) &::= (z_0^{b \rightarrow b} \circ z_1^{b \rightarrow b}) \sqcap (y_0^{b \rightarrow b} \circ y_1^{b \rightarrow b}) \\
f(\neg \phi_1^3) &::= (y_0^{b \rightarrow b} \sqcap y_1^{b \rightarrow b}) \circ (z_0^{b \rightarrow b} \sqcap z_1^{b \rightarrow b}) \\
f(\phi_2^3) &::= (z_0^{b \rightarrow b} \circ z_1^{b \rightarrow b} \circ z_2^{b \rightarrow b}) \sqcap ((y_0^{b \rightarrow b} \sqcap y_1^{b \rightarrow b}) \circ y_2^{b \rightarrow b}) \\
f(\neg \phi_2^3) &::= (y_0^{b \rightarrow b} \sqcap y_1^{b \rightarrow b} \sqcap y_2^{b \rightarrow b}) \circ ((z_0^{b \rightarrow b} \circ z_1^{b \rightarrow b}) \sqcap z_2^{b \rightarrow b})
\end{aligned}$$

5 Open Problems

In this paper, we have presented and analyzed the 2EC evaluation problem. We have concluded that 2EC (and therefore second-order conjunctive strictness analysis) is *NP*-hard. This result obviously leaves many open problems left to examine. There are two very important open questions about 2EC worth noting.

5.1 Deterministic Exponential Time Algorithm?

Since we have not presented an algorithm to solve 2EC, we cannot comment on the “upper bound” time complexity of 2EC. Since 2E can be solved in “double-exponential” time, we know 2EC can be, too. However, it seems possible that the 2EC problem can be solved with a deterministic exponential time algorithm.

5.2 Deterministic Exponential Hard?

The idea of using first-order functions to represent Boolean values suggests that the 1E problem can be poly-time many-one reduced to 2EC. Specifically:

- We “simulate” the value \perp with the first-order function I.
- We “simulate” the value \top with the first-order function T.
- We “simulate” the lub operator \sqcup with the composition of two functions.

Informally, we illustrate the above idea with an example. Consider the following *depth* 1 RMS $S = ((x_1^{b^3 \rightarrow b}, e_1), (x_2^{b^2 \rightarrow b}, e_2), (x_3^b, e_3))$, where:

$$\begin{aligned} e_1 &::= \lambda y_1^b y_2^b y_3^b. \\ &\quad [y_1^b \sqcup (y_2^b \sqcap x_2^{b^2 \rightarrow b}(y_3^b, y_3^b))] \\ e_2 &::= \lambda y_1^b y_2^b. \\ &\quad [y_2^b \sqcup x_1^{b^3 \rightarrow b}(y_1^b \sqcup y_2^b, y_2^b, y_1^b)] \\ e_3 &::= x_2^{b^2 \rightarrow b}(\top^b, \perp^b) \end{aligned}$$

We reduce S to the following *depth* 2 RCMS:

$$S' = ((x_1^{(b \rightarrow b)^3 \rightarrow (b \rightarrow b)}, e'_1), (x_2^{(b \rightarrow b)^2 \rightarrow (b \rightarrow b)}, e'_2), (x_3^{b \rightarrow b}, e'_3), (x_4^b, e'_4))$$

where:

$$\begin{aligned} e'_1 &::= \lambda y_1^{b \rightarrow b} y_2^{b \rightarrow b} y_3^{b \rightarrow b}. \\ &\quad [y_1^{b \rightarrow b} \circ (y_2^{b \rightarrow b} \sqcap x_2^{(b \rightarrow b)^2 \rightarrow (b \rightarrow b)}(y_3^{b \rightarrow b}, y_3^{b \rightarrow b}))] \\ e'_2 &::= \lambda y_1^{b \rightarrow b} y_2^{b \rightarrow b}. \\ &\quad [y_2^{b \rightarrow b} \circ x_1^{(b \rightarrow b)^3 \rightarrow (b \rightarrow b)}(y_1^{b \rightarrow b} \circ y_2^{b \rightarrow b}, y_2^{b \rightarrow b}, y_1^{b \rightarrow b})] \\ e'_3 &::= x_2^{(b \rightarrow b)^2 \rightarrow (b \rightarrow b)}(\lambda x^b. \top^b, \lambda x^b. x^b) \\ e'_4 &::= x_3^{b \rightarrow b}(\perp^b) \end{aligned}$$

It would seem that $\llbracket x_3^b \rrbracket \rho_S = \top$ iff $\llbracket x_4^b \rrbracket \rho_{S'} = \top$.

Unfortunately, this straightforward reduction is incorrect. In some cases of *depth* 1 RMS's, the resulting *depth* 2 RCMS S' will not "behave" properly. Specifically, the problem lies within the specification of least-fixpoints in the environment $\rho_{S'}$. Because of recursion, $\rho_{S'}$ sometimes map terms to the constant \perp function from $\mathbf{2}$ to $\mathbf{2}$, when we would like the terms to be mapped to either I or T. (In fact, in the proof of Theorem 4.1, we see that $\llbracket x_2^{b \rightarrow b} \rrbracket \rho_{S_\phi}$ maps to the constant \perp function instead of I, when ϕ is not satisfiable.) This will cause complications in the reduction.

For example, let $S = ((x_1^{b^2 \rightarrow b}, e_1), (x_2^b, e_2))$, where:

$$\begin{aligned} e_1 &::= \lambda y_1^b y_2^b. [x_1^{b^2 \rightarrow b}(y_2^b, y_1^b) \sqcup y_1^b \sqcup x_1^{b^2 \rightarrow b}(y_1^b, y_1^b \sqcap y_2^b)] \\ e_2 &::= x_1^{b^2 \rightarrow b}(\top^b, \top^b) \end{aligned}$$

Then $S' = ((x_1^{(b \rightarrow b)^2 \rightarrow (b \rightarrow b)}, e'_1), (x_2^{b \rightarrow b}, e'_2), (x_3^b, e'_3))$ is the *depth* 2 RCMS defined by the reduction, where:

$$\begin{aligned} e'_1 &::= \lambda y_1^{b \rightarrow b} y_2^{b \rightarrow b}. \\ &\quad x_1^{(b \rightarrow b)^2 \rightarrow (b \rightarrow b)}(y_2^{b \rightarrow b}, y_1^{b \rightarrow b}) \circ y_1^{b \rightarrow b} \circ \\ &\quad x_1^{(b \rightarrow b)^2 \rightarrow (b \rightarrow b)}(y_1^{b \rightarrow b}, y_1^{b \rightarrow b} \sqcap y_2^{b \rightarrow b}) \\ e'_2 &::= x_1^{(b \rightarrow b)^2 \rightarrow (b \rightarrow b)}(\lambda x^b. \top^b, \lambda x^b. \top^b) \\ e'_3 &::= x_2(\perp^b) \end{aligned}$$

It is clear that $\llbracket x_2^b \rrbracket \rho_S = \top$. However, since $\rho_{S'}$ must specify least-fixpoints:

$$\llbracket x_1^{(b \rightarrow b)^2 \rightarrow (b \rightarrow b)} \rrbracket \rho_{S'} = \perp_{(\mathbf{2} \rightarrow \mathbf{2})^2 \rightarrow (\mathbf{2} \rightarrow \mathbf{2})}$$

This means that $\llbracket x_3^b \rrbracket \rho_{S'} = \perp$. Thus the above reduction does not hold.

It is the opinion of the author that such problems may be correctable, but more work must be done in order to do so.

6 Acknowledgements

I would like to express my gratitude for Prof. Albert Meyer for his guidance in supervising my research. I would also like to thank Jon Riecke for his help in understanding "least fixpoints".

References

- [1] G.I. Burn, C.L. Hankin, and S. Abramsky. *The theory and practice of strictness analysis for higher order functions*. Technical Report DoC 85/6, Dept. of Computing, Imperial College of Science and Technology, London SW7 2BZ, Great Britain, April 1985.
- [2] C. Clack and S. L. Peyton Jones. Strictness analysis - a practical approach. In *Functional Programming Languages and Computer Architecture*, Springer-Verlag LNCS 201, September 1985.
- [3] Joost Engelfriet. Iterated pushdown automata and complexity classes. In *15th Symp. Theory of Computation*, pages 365-373, ACM, 1983.
- [4] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.
- [5] Paul Hudak and Jonathan Young. Finding fixpoints on function spaces. December 1986. Unpublished manuscript, Yale University.
- [6] Paul Hudak and Jonathan Young. Higher-order strictness analysis in untyped lambda calculus. In *13th Symp. Principles of Programming Languages*, pages 97-109, ACM, 1986.
- [7] Christos Kaklamanis. *A Special Case of First-Order Strictness Analysis*. Bachelors Thesis, MIT, 1986.
- [8] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, Inc., 1981.
- [9] A. Mycroft. *Abstract Interpretation and Optimising Transformations for Applicative Programs*. PhD thesis, Univ. of Edinburgh, 1981.
- [10] F. Nielson. Strictness analysis and denotational abstract interpretation. *Information and Control*, 76:29-92, 1988.
- [11] H. Seidl. Parameter-reduction of higher level grammars. *Theoretical Computer Science*, 55:47-88, November 1987.

- [12] R. Statman. The typed λ -calculus is not elementary recursive. *Theoretical Computer Science*, 9:73–81, 1979.
- [13] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

OFFICIAL DISTRIBUTION LIST

Director Information Processing Techniques Office Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209	2 copies
Office of Naval Research 800 North Quincy Street Arlington, VA 22217 Attn: Dr. R. Grafton, Code 433	2 copies
Director, Code 2627 Naval Research Laboratory Washington, DC 20375	6 copies
Defense Technical Information Center Cameron Station Alexandria, VA 22314	12 copies
National Science Foundation Office of Computing Activities 1800 G. Street, N.W. Washington, DC 20550 Attn: Program Director	2 copies
Dr. E.B. Royce, Code 38 Head, Research Department Naval Weapons Center China Lake, CA 93555	1 copy